Prototyping Games using Formal Methods

Sebastian Krings, Philipp Körner

Niederrhein University of Applied Sciences, University of Düsseldorf

Hochschule Niederrhein
University of Applied Sciences

Examples in FM Courses

- Supposed to show students how to apply formal methods
- Often one of two kinds:
  - Quite artificial and unrelated to practice
  - Based on projects of industry partners and way too involved for students

Games to the Rescue

We deem games more suited as examples:

- Well-known to the students
  - Can focus on modeling, proving and methodology, rather than intended properties
  - Reduces requirements engineering
- Modern computer games are very sophisticated
  - Use of formal methods appropriate
- Allow to challenge our tools and thus drive research

Methods & Tools Used

- B Method, both classical and Event-B
- Tools
  - ProB and ProB 2.0
  - Rodin
  - BMotionWeb

Case Studies

- Pac-Man
- Chess
- LightBot

Pac-Man Requirements

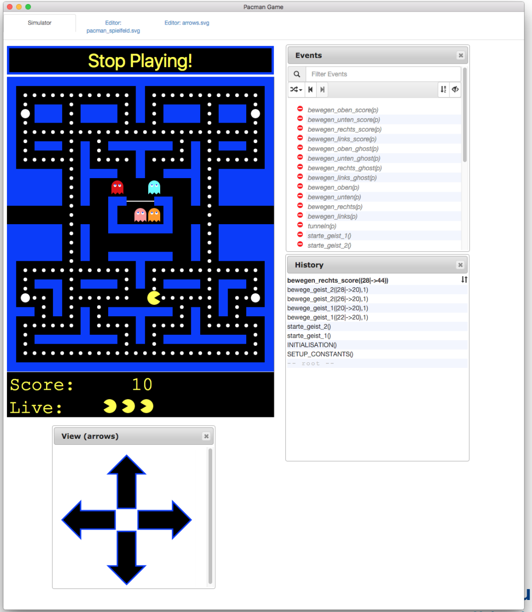
1. Pac-Man can only be moved from one field of the grid to a direct neighbor field. It cannot jump.
2. The same holds for ghosts.
3. Pac-Man can only be moved when every ghost, that must have been started, has moved at least once after the last movement of Pac-Man.
4. Pac-Man can be moved through a tunnel.
5. The first two ghosts must start before Pac-Man starts.
6. The third / fourth ghost must start after 30 / 180 collected dots.
7. Each dot can only be collected once.
8. If Pac-Man and a ghost share positions, one catches the other.
9. If a ghost catches Pac-Man, the player loses a life.

Pac-Man Implementation Detail

Different refinements and implementation detail for students to experiment with:

- board representation
- ensure movement as expected
- ensure order of movement (invariant? LTL?)
- how to represent continuous movement in state-based method
- step sizes, i.e., what should count as a state transition
- properly start / stop the game

Pac-Man Visualization



Pac-Man Demo

Demo Video

Pac-Man drives FM development

Pac-Man also served as a playground for novel research directions:

- Can the prototypical model made playable without further code generation
- Experiment with state-space search algorithms beyond simple depth-first or breath-first traversal.

Chess Requirements

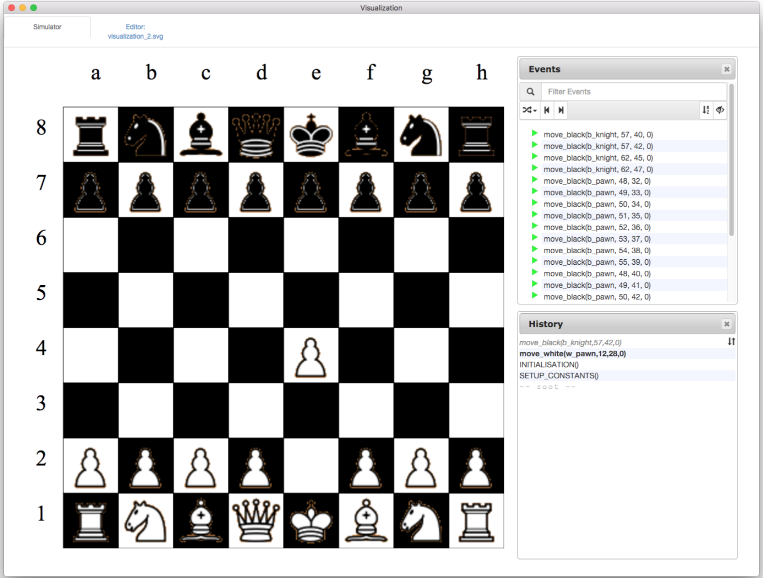
1. Pieces can only be moved in their specific way (e. g., a king can only move exactly one field into any direction).
2. If the king is in check, only moves getting the king out of check are permitted.
3. No piece can be moved outside the 8×8 board.
4. Special moves (Castling, En Passant and Promotion) follow the rules.
5. If the king cannot be defended immediately, the game is lost.
6. If no legal move is possible for one player, the game is considered as a draw.
7. Both players have the same set of pieces and the white player has the first move.

Chess Implementation Detail

Again, lots of issues to experiment with:

- piece-centric vs. field-centric representation
- special moves en passant, castling, etc.
- exchange pawns
- evaluate quality of position
- ...

Chess Visualization



Chess Demo

Live Demo + Part of Lab

Chess drives FM development

Minimax as model checking heuristic to control state space exploration:

- Values of figures residing on the board following S. E. Claude
- Pawns in desired or undesired positions, e.g., passed pawns
- Number of semi-open files, i.e., the number of rows or columns the player's rooks can move at least five fields into one direction on.
- Count how well the fields adjacent to the own king are guarded, again applying a weight of 2.
- Measure to what extent a player controls the four squares in the center. As they are usually crucial to winning the game, we apply a weight of 3.

LightBot

An educational game on programming:

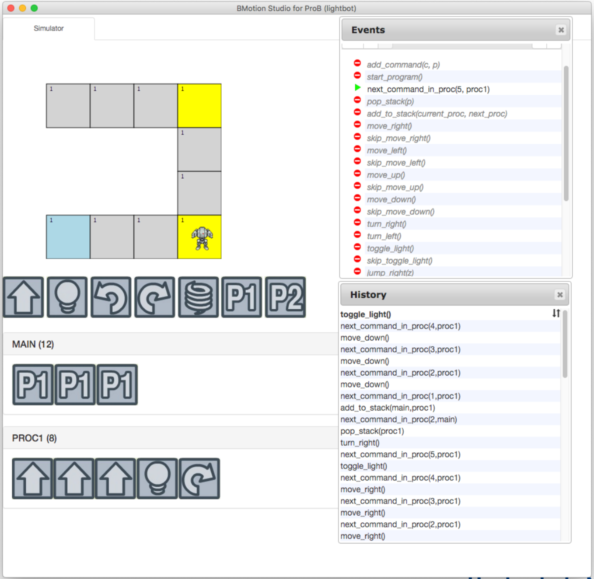
- Player has to program a robot to turn on lights
- Can use if, ... define sub-routines, ...
- Restrictions on code increase in higher levels

$\Rightarrow$ game is an interpreter and we specify it!

LightBot Requirements

1. The robot moves on a three-dimensional board.
2. The game is generic, i.e., different levels (boards) are supported and can be provided and switched in some way.
3. The robot supports all moves (forward, toggle light, left/right turn, jumping and entering one of two sub-procedures).
4. The robot starts execution in the main-procedure.
5. A program stack is required to execute the user-defined sub-routines, as the may be mutually recursive. Again, this underlines the idea that students do in fact specify the internal workings of an interpreter.
6. The lowest elevation level is 1.
7. Starting position and the tiles the robot has to light up to complete the level are described in the level itself, not hard-coded in the interpreter.

LightBot Visualization



LightBot drives FM development

- Original game is an educational game on coding.
- Used to teach basic programming concepts, such as function calls, recursion and loops.
- Following this idea, writing a specification of the game itself (as opposed to a specification of the player-given code to solve a level) teaches how to *model* and *verify* function calls, recursion ...
- Students learn how to model programming languages and their interpreters.
- The same concept could later be applied to “real” programming languages with more sophisticated semantics.

Conclusion: Tools

ProB

- (Bounded) model checking gives fast feedback
- Animation, in particular including visualization on top, allows reassuring students that changes behave as intended.
- Sometimes cannot cope with the state spaces of games

Conclusion: Tools

Student feedback concerning Rodin is rather negative:

- Usability is lacking
- Sometimes in an inconsistent state
- Machines are not plain text, structural editors are default. Students find it uncomfortable to switch between text boxes.
- Furthermore, some functions are hidden in context menus that only pop up when right-clicking on very specific positions.
- Finally, the files do not integrate well with version control.

Conclusion: Tools

BMotionWeb

- Great to explain specifications to students
- Application based entirely on web technologies is hard to use
- When errors occur, it is not clear where the cause is located: is it an error in the B model? Is an SVG file broken? Is the config file incorrect?

Conclusions: Impact on Learning

- Hard to measure influence on interest, attention and understanding
- No clear trend that correlates with games as examples: overall student feedback remained the same
- Grades improved significantly after introducing mandatory projects based on Lightbot
- In the following years, grades worsened without changing anything
- Upon introduction of other examples, grades improved again
- Games definitely improved engagement

But why?

- Breaking the routine of the teaching personnel is more engaging for students?
- Not everybody likes games as much?
- Some versions of the projects were shared between students over years, and parts where copied, resulting in students missing crucial learning outcomes?

Conclusions: Overall

- FM can be applied to game prototypes
- Have proven properties about game implementations and the correct representation of the rules of a game
- Playability is limited, continuous movement hard
- Games make for easy to understand and highly motivating examples for students
- Turn-based games are a great match and can be fun and engaging to interact with
- While performance less critical for teaching it limits applicability of FM to games

Thank you!

Any questions?